Programación de Microservicios con Spring Boot y Red Hat SSO

Nivel Avanzado

Instructor: Carlos Carreño
ccarrenovi@gmail.com

Modulo 3. El API Gateway

Objetivo: Aprender a utilizar API Gateway en una arquitectura de microservicios para gestionar rutas, seguridad y balanceo de carga.

Duración: 1.5 h

Igp | Innovación Gestión Buenas Prácticas

Configurar Zuul con Spring Boot

Zuul es un **API Gateway** desarrollado por Netflix que permite:

- Ruteo dinámico a microservicios
- Filtros pre/post de petición
- Control de acceso y seguridad
- Balanceo de carga con Ribbon (obsoleto)

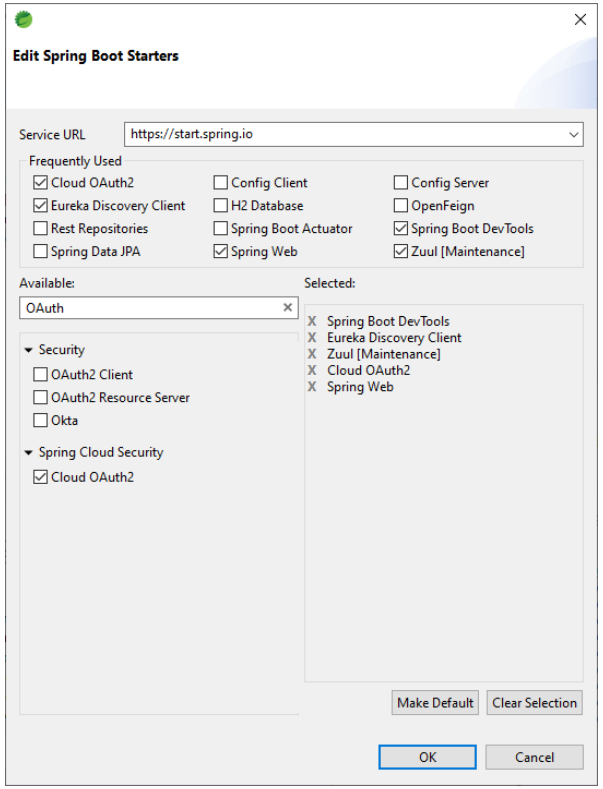
Zuul vs Spring Cloud Gateway

- Comparación

Característica	Zuul (1.x)	Spring Cloud Gateway
Soporte	Obsoleto	Activo y recomendado
Basado en	Servlet (Tomcat)	Reactor (WebFlux)
Mejor rendimiento	❌	✅
Filtros programáticos	Verbosos	Funcionales y reactivos
Soporta WebSockets	❌ (limitado)	✅

Implementando Zuul y Microservicios

- Debes agregar la dependencia en el proyecto



Seguridad de Zuul con JWT

- Se debe proteger el **Zuul Gateway** para que solo acepte peticiones con un JWT válido. Usaremos un **filtro pre** para interceptar las peticiones y verificar el token.

```
@Component
public class JwtAuthorizationFilter extends ZuulFilter {

    private final String SECRET = "secreto123"; // usa algo más seguro en producción

    @Override
    public String filterType() {
        return "pre"; // filtro antes de enrutar
    }

    @Override
    public int filterOrder() {
        return 1;
    }
```

Balanceo de carga con Zuul

Para que **Zuul realice balanceo de carga**, se apoya en:

- **Eureka** (Service Discovery)
- **Ribbon** (cliente de balanceo de carga, integrado en Spring Cloud)

Spring Cloud Gateway

- Es una **puerta de entrada (API Gateway)** que maneja el enrutamiento, filtros, seguridad, balanceo de carga, CORS, etc.
- Fue diseñado para ser más rápido y reactivo que Zuul, usando WebFlux (reactivo, no bloqueante).

Componente	Descripción
Route	Define cómo enrutar solicitudes entrantes a destinos
Predicate	Condiciones para que una ruta sea aplicada (por ejemplo, por path, método, header, etc.)
Filter	Permite modificar la solicitud o respuesta (como añadir cabeceras, logs, etc.)
URI	Destino final (servicio de backend o URL externa)

Spring Cloud Gateway: Ejemplo

- Esto enruta peticiones a **/cliente/**** hacia **http://localhost:8081** eliminando el prefijo **/cliente**.

```yaml
server:
  port: 8080

spring:
  cloud:
    gateway:
      routes:
        - id: cliente-service
          uri: http://localhost:8081
          predicates:
            - Path=/cliente/**
          filters:
            - StripPrefix=1
```

curl http://localhost:8080/cliente/mensaje Se redirige a **http://localhost:8081/mensaje**

Spring Cloud Gateway: Balanceo de Carga

- Para que el Gateway descubra y balancee carga entre múltiples instancias, solo necesitas:
- Agregar las dependencias

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

...

- Define el esquema de load Balancer. Usa el esquema lb:// para que use Ribbon + Eureka

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: cliente
          uri: lb://cliente-service
          predicates:
            - Path=/cliente/**
          filters:
            - StripPrefix=1

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```